DAC_Phase5 : Water Quality Analysis Project

Introduction:

The goal of the "Water Quality Analysis Project" in Phase 3, is to perform preprocessing and Exploratory Data Analysis by plotting graphs and getting insights.

Our approach involves,

```
   1. finding correlation between the attributes of the dataset
provided,
   2. Handling missing values,
   3. Getting comparative insights by using necessary plots for
further processing and clear understanding on dataset attributes.
```

Analysis Objectives:

1. Protection of Human Health: One of the primary objectives of water quality analysis is to ensure that water is safe for human consumption. We can assess whether the water meets health-based standards and guidelines by monitoring various parameters such as microbial contaminants, chemical pollutants, and physical characteristics.
2. Protection of Aquatic Ecosystems: Water quality analysis aims to maintain and improve the health of aquatic ecosystems. By assessing factors like nutrient levels, dissolved oxygen, and pH, we can identify potential threats to aquatic life and take corrective measures.
3. Resource Management: Water quality analysis helps in managing water resources effectively. It provides information on water availability, suitability for different uses (e.g., drinking, agriculture, industry), and potential risks to these resources.
4. Compliance with Regulations: Governments and environmental agencies set water quality standards and regulations. Water quality analysis ensures compliance with these standards by monitoring pollutants and other relevant parameters.
5. Early Detection of Pollution: Regular monitoring allows us to detect pollution sources early. By identifying changes in water quality trends, we can take timely actions to prevent further degradation.
6. Assessment of Pollution Sources: Water quality analysis helps pinpoint pollution sources (e.g., industrial discharges, agricultural runoff) by analyzing specific contaminants or pollutants.
7. Baseline Data: Establishing baseline water quality data provides a reference point for future assessments. It allows us to track changes over time and evaluate the effectiveness of pollution control measures.
8. Transboundary Cooperation: In transboundary waters shared by multiple countries, water quality analysis facilitates cooperation in setting joint objectives and criteria for maintaining water quality across borders

Python Libraries

```
#importing necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

Reading DataseIm

Importing Dataset: https://www.kaggle.com/datasets/adityakadiwal/water-potability

```
# Creating DataFrame by using .csv file
df = pd.read_csv("archive/water_potability.csv")

df.head()
```

```
         ph    Hardness        Solids  Chloramines     Sulfate
Conductivity \
0       NaN  204.890455  20791.318981     7.300212  368.516441
564.308654
1  3.716080  129.422921  18630.057858     6.635246         NaN
592.885359
2  8.099124  224.236259  19909.541732     9.275884         NaN
418.606213
3  8.316766  214.373394  22018.417441     8.059332  356.886136
363.266516
4  9.092223  181.101509  17978.986339     6.546600  310.135738
```

```
398.410813
```

```
   Organic_carbon  Trihalomethanes  Turbidity  Potability
0       10.379783        86.990970   2.963135           0
1       15.180013        56.329076   4.500656           0
2       16.868637        66.420093   3.055934           0
3       18.436524       100.341674   4.628771           0
4       11.558279        31.997993   4.075075           0
```

```
# Descriptive Statistics
df.describe()
```

```
                ph     Hardness        Solids  Chloramines
Sulfate \
count  2785.000000  3276.000000   3276.000000  3276.000000
2495.000000
mean      7.080795   196.369496  22014.092526     7.122277
333.775777
std       1.594320    32.879761   8768.570828     1.583085
41.416840
min       0.000000    47.432000    320.942611     0.352000
129.000000
25%       6.093092   176.850538  15666.690297     6.127421
307.699498
50%       7.036752   196.967627  20927.833607     7.130299
333.073546
75%       8.062066   216.667456  27332.762127     8.114887
359.950170
max      14.000000   323.124000  61227.196008    13.127000
481.030642
```

```
       Conductivity  Organic_carbon  Trihalomethanes    Turbidity
Potability
count   3276.000000     3276.000000      3114.000000  3276.000000
3276.000000
mean     426.205111       14.284970        66.396293     3.966786
0.390110
std       80.824064        3.308162        16.175008     0.780382
0.487849
min      181.483754        2.200000         0.738000     1.450000
0.000000
25%      365.734414       12.065801        55.844536     3.439711
0.000000
50%      421.884968       14.218338        66.622485     3.955028
0.000000
75%      481.792304       16.557652        77.337473     4.500320
1.000000
max      753.342620       28.300000       124.000000     6.739000
1.000000
```

```
# Information about dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

Correlation Between Features

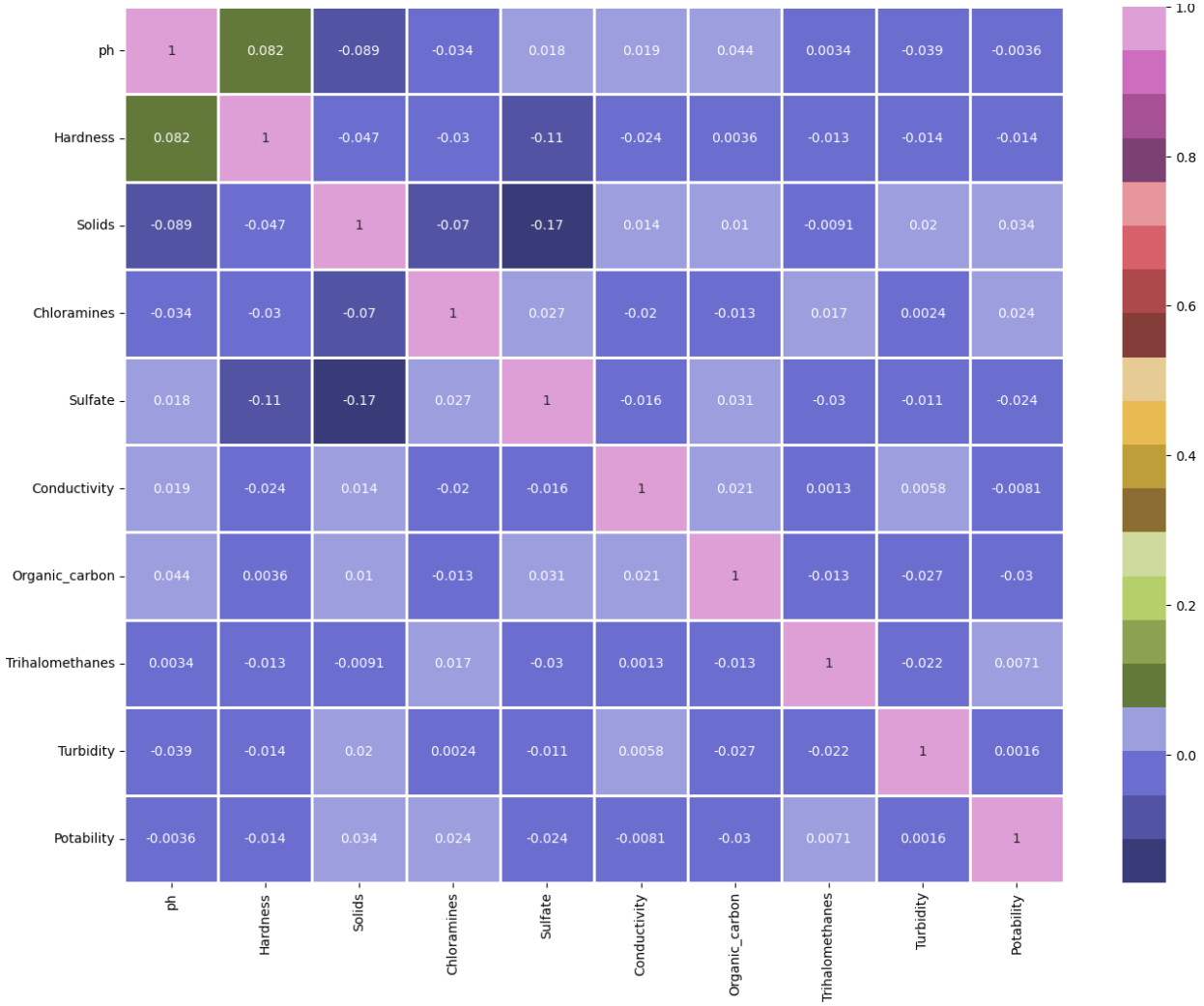
```
#correlation table
df.corr()
```

```
                      ph  Hardness    Solids Chloramines  
Sulfate \
ph                 1.000000  0.082096 -0.089288    -0.034350  0.018203

Hardness           0.082096  1.000000 -0.046899    -0.030054 -0.106923

Solids            -0.089288 -0.046899  1.000000    -0.070148 -0.171804

Chloramines       -0.034350 -0.030054 -0.070148     1.000000  0.027244

Sulfate            0.018203 -0.106923 -0.171804     0.027244  1.000000

Conductivity       0.018614 -0.023915  0.013831    -0.020486 -0.016121

Organic_carbon     0.043503  0.003610  0.010242    -0.012653  0.030831

Trihalomethanes    0.003354 -0.013013 -0.009143     0.017084 -0.030274

Turbidity         -0.039057 -0.014449  0.019546     0.002363 -0.011187

Potability        -0.003556 -0.013837  0.033743     0.023779 -0.023577
```

```
                 Conductivity  Organic_carbon  Trihalomethanes
Turbidity  \
ph                   0.018614        0.043503         0.003354  -
0.039057
Hardness            -0.023915        0.003610        -0.013013  -
0.014449
Solids               0.013831        0.010242        -0.009143
0.019546
Chloramines         -0.020486       -0.012653         0.017084
0.002363
Sulfate             -0.016121        0.030831        -0.030274  -
0.011187
Conductivity         1.000000        0.020966         0.001285
0.005798
Organic_carbon       0.020966        1.000000        -0.013274  -
0.027308
Trihalomethanes      0.001285       -0.013274         1.000000  -
0.022145
Turbidity            0.005798       -0.027308        -0.022145
1.000000
Potability          -0.008128       -0.030001         0.007130
0.001581
                 Potability
ph                -0.003556
Hardness          -0.013837
Solids             0.033743
Chloramines        0.023779
Sulfate           -0.023577
Conductivity      -0.008128
Organic_carbon    -0.030001
Trihalomethanes    0.007130
Turbidity          0.001581
Potability         1.000000
```

```
#correlation by using clustermap
#sns.heatmap(df.corr(), cmap='flag')

fig, ax = plt.subplots(figsize=(16, 12))
sns.heatmap(df.corr(),
cmap='tab20b',annot=True,linewidths='0.8',ax=ax)
```

```
<Axes: >
```



Preprocessing: Missing Value

```
#missing value counts
df.isnull().sum()
```

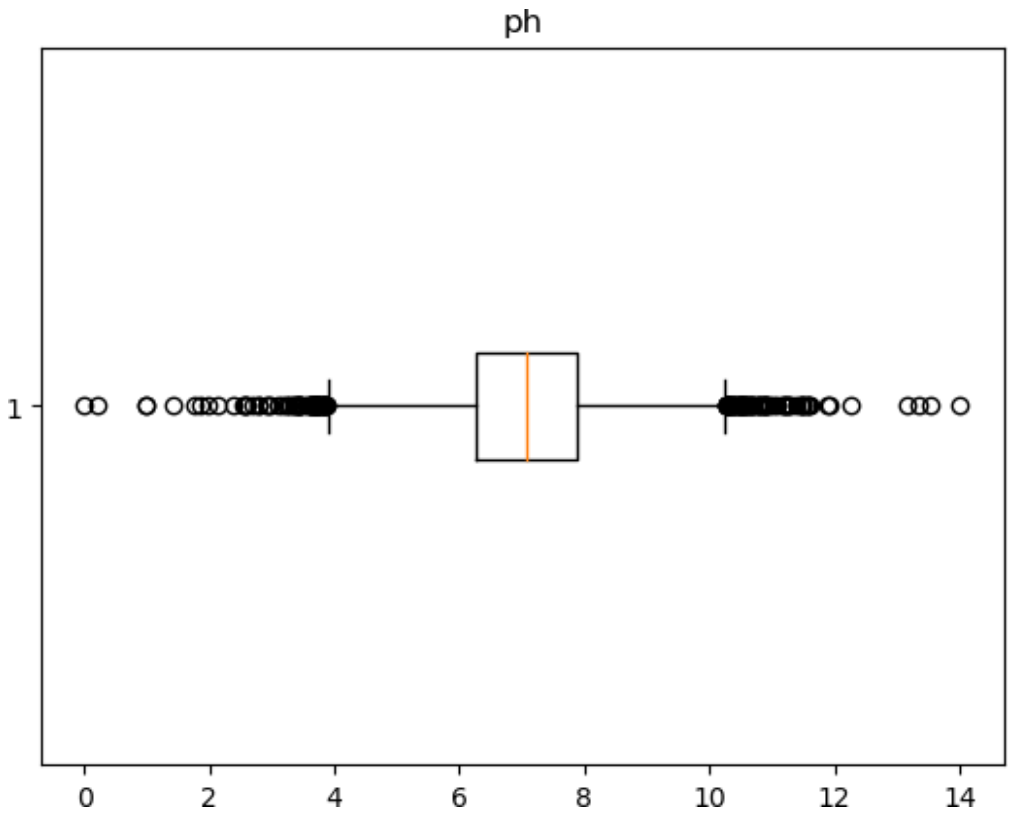
```
ph                  491
Hardness              0
Solids                0
Chloramines           0
Sulfate             781
Conductivity          0
Organic_carbon        0
Trihalomethanes     162
Turbidity             0
Potability            0
dtype: int64
```

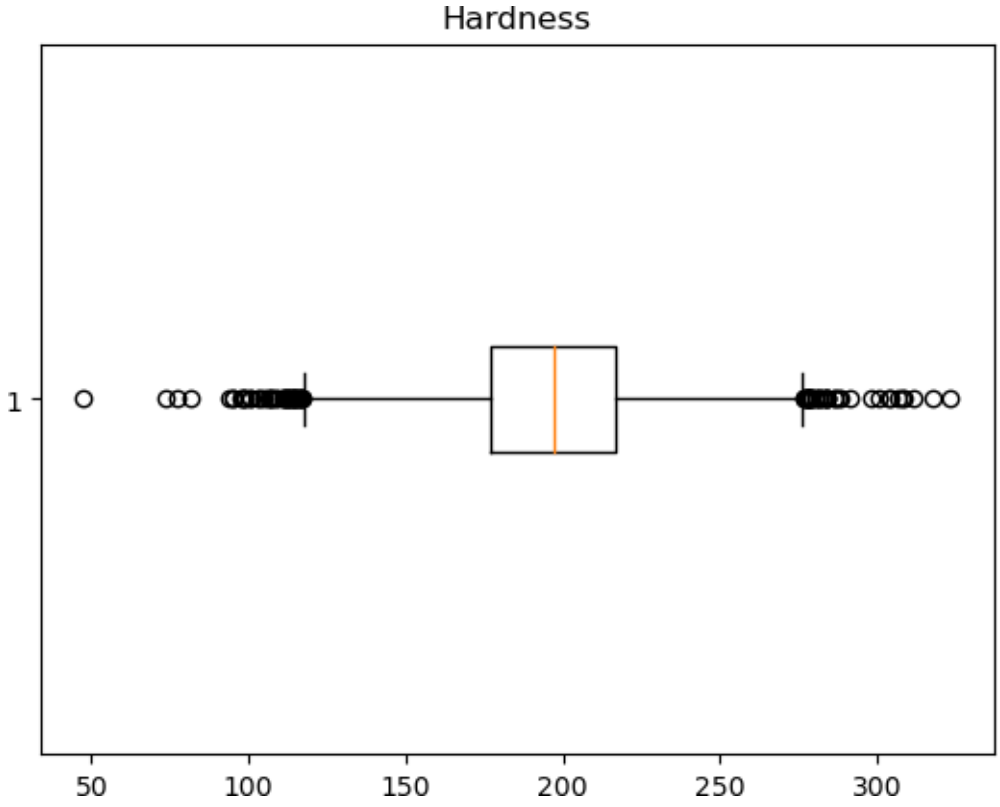
```
df['ph'].fillna(value = df['ph'].mean(), inplace = True)

df['Sulfate'].fillna(value = df['Sulfate'].mean(), inplace = True)
df['Trihalomethanes'].fillna(value = df['Trihalomethanes'].mean(),
inplace = True)

# Check again the missing values
df.isnull().sum()
```

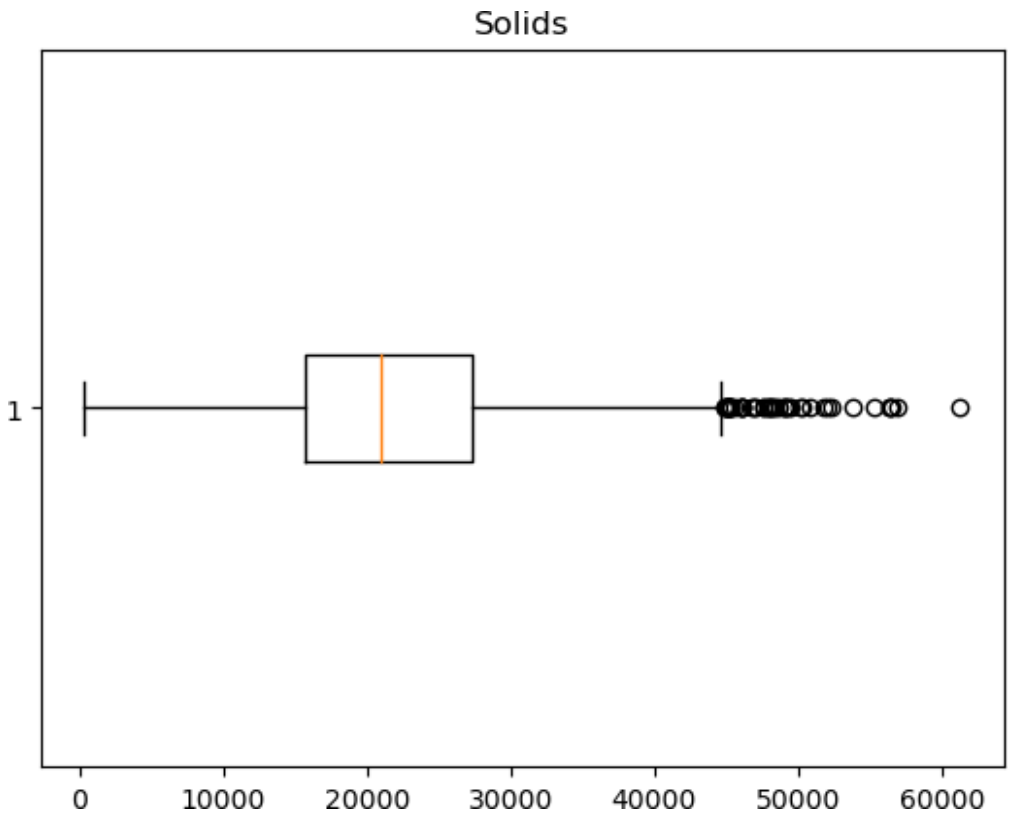
```
ph                  0
Hardness            0
Solids              0
Chloramines         0
Sulfate             0
Conductivity        0
Organic_carbon      0
Trihalomethanes     0
Turbidity           0
Potability          0
dtype: int64
```

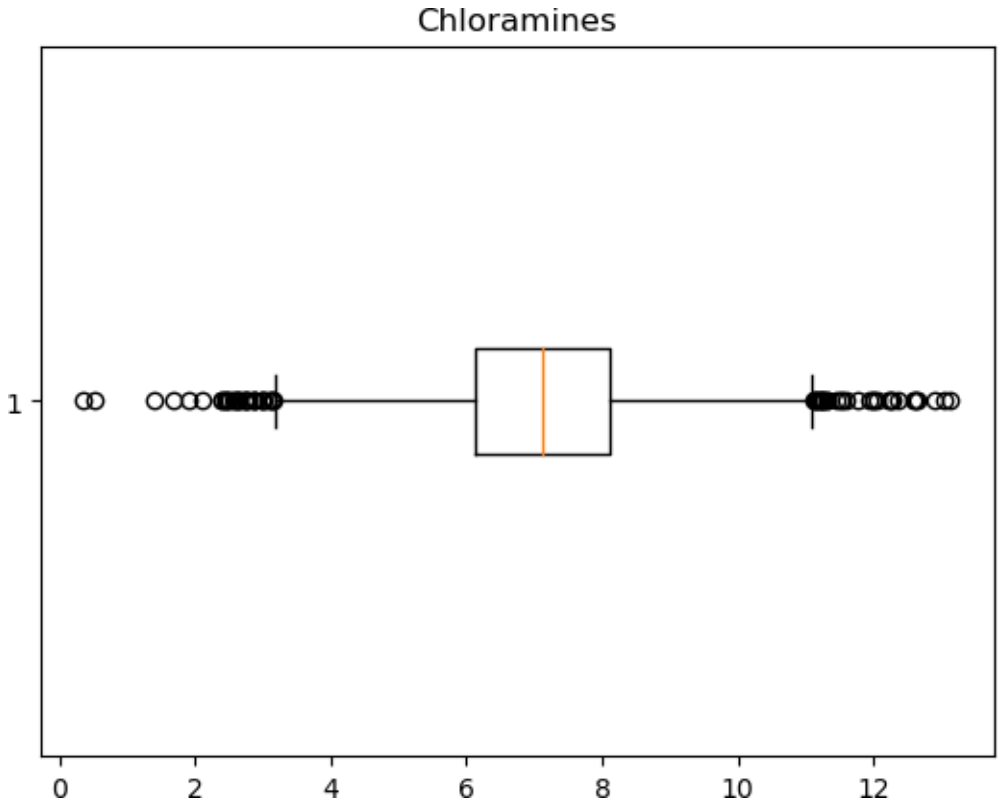
Checking for outliers using boxplot

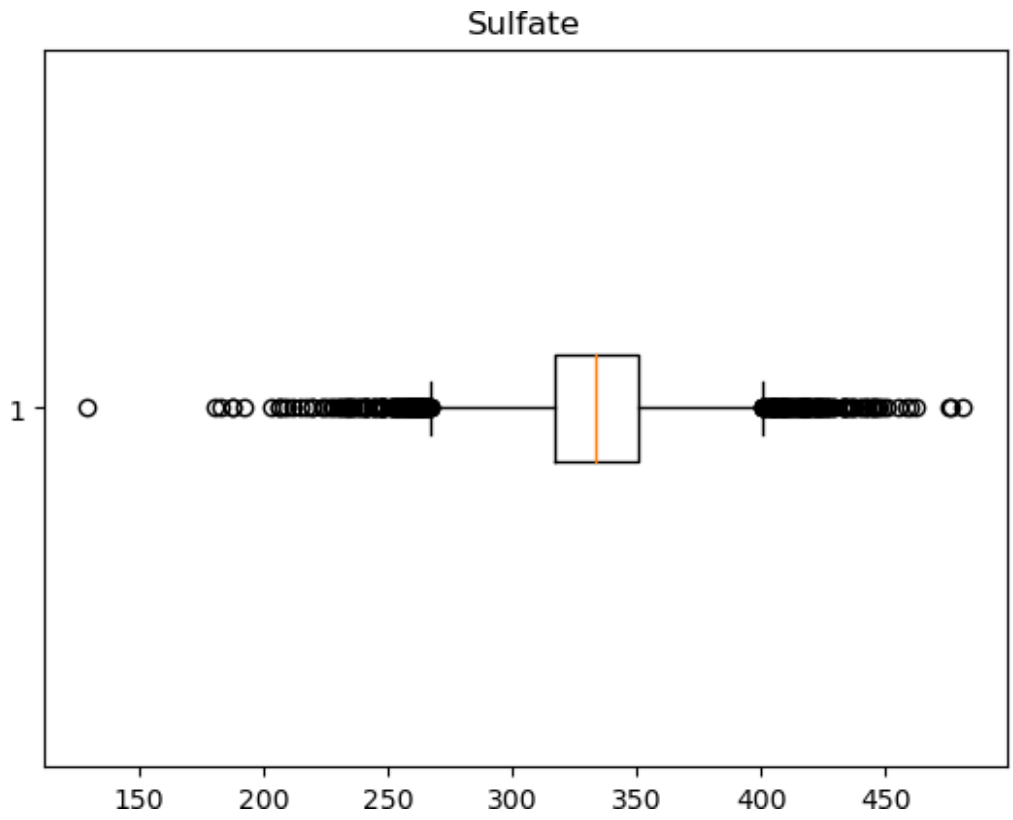
```
for col in df.columns:
    plt.boxplot(df[col], vert=False)
    plt.title(col)
    plt.show()
```

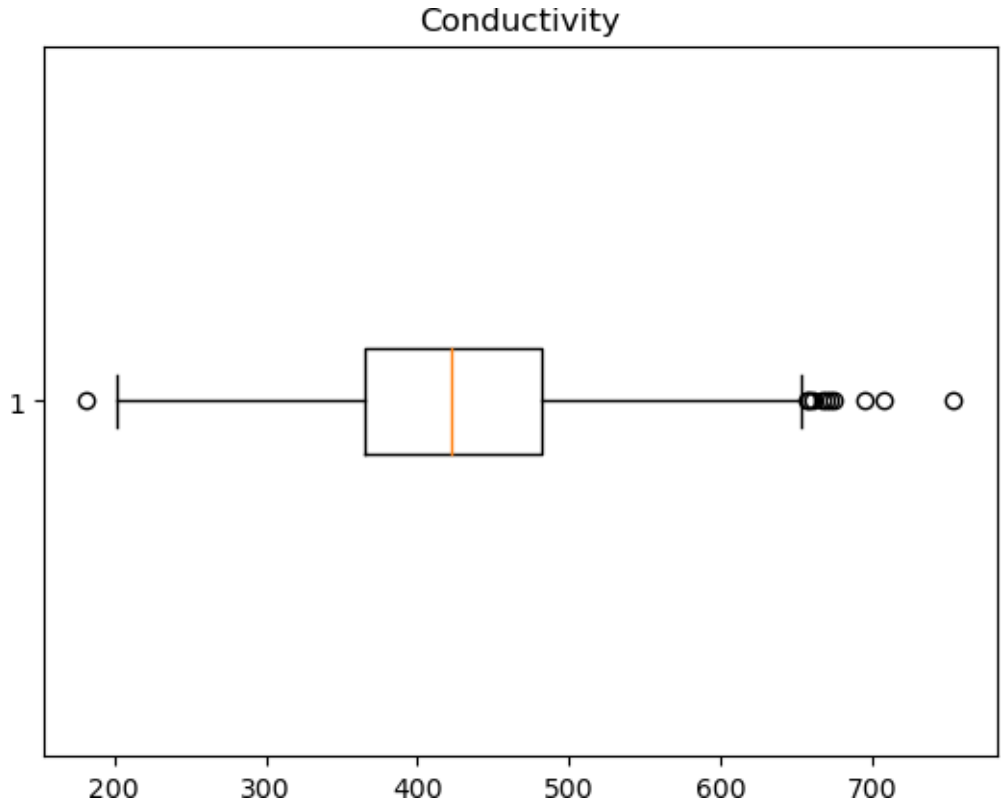

ph
1
0
2
4
6
8
10
12
14

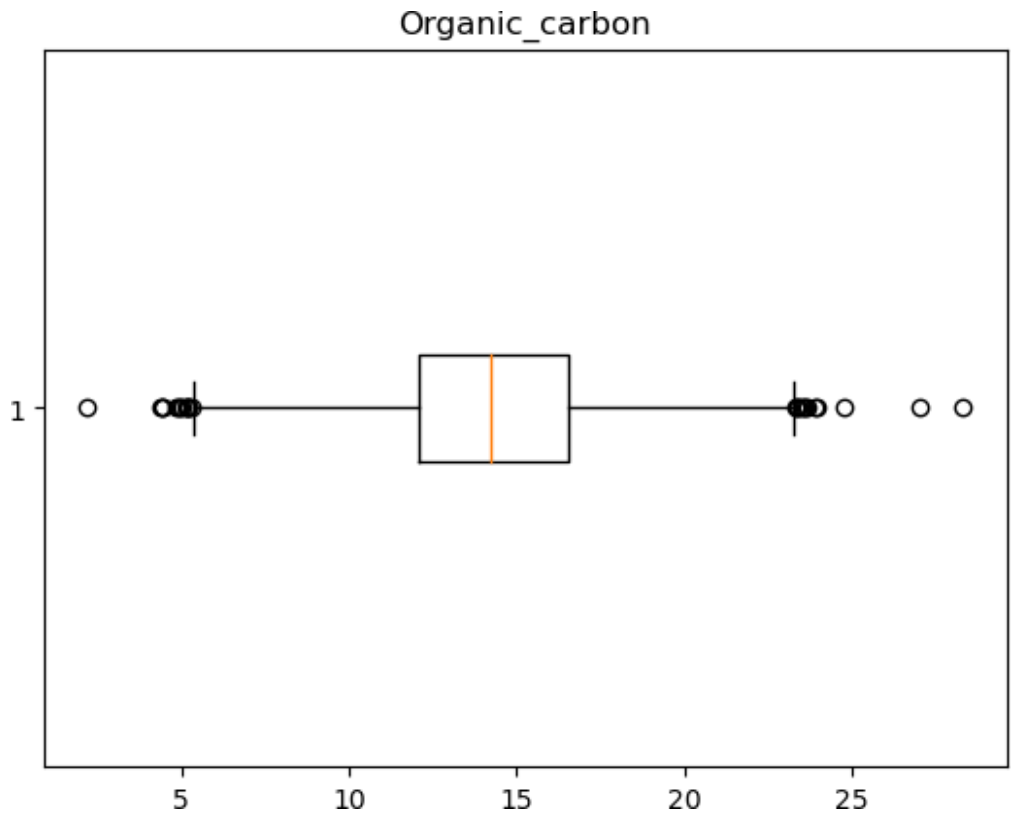

Hardness
1
50
100
150
200
250
300

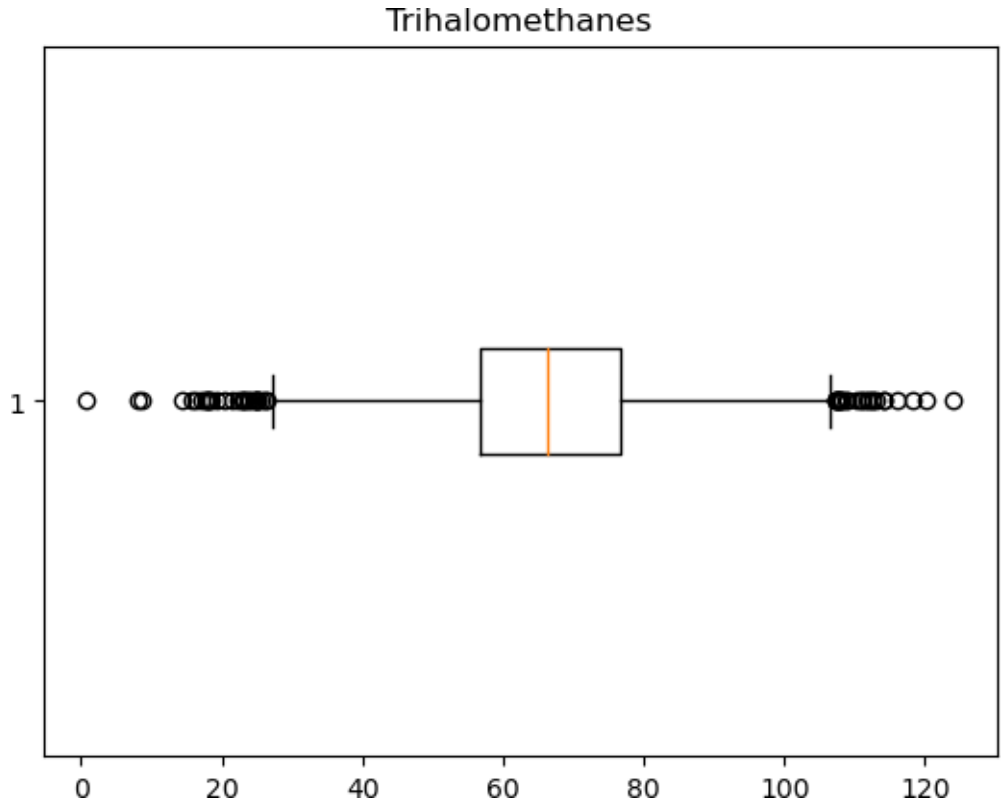

Solids
1
0
10000
20000
30000
40000
50000
60000

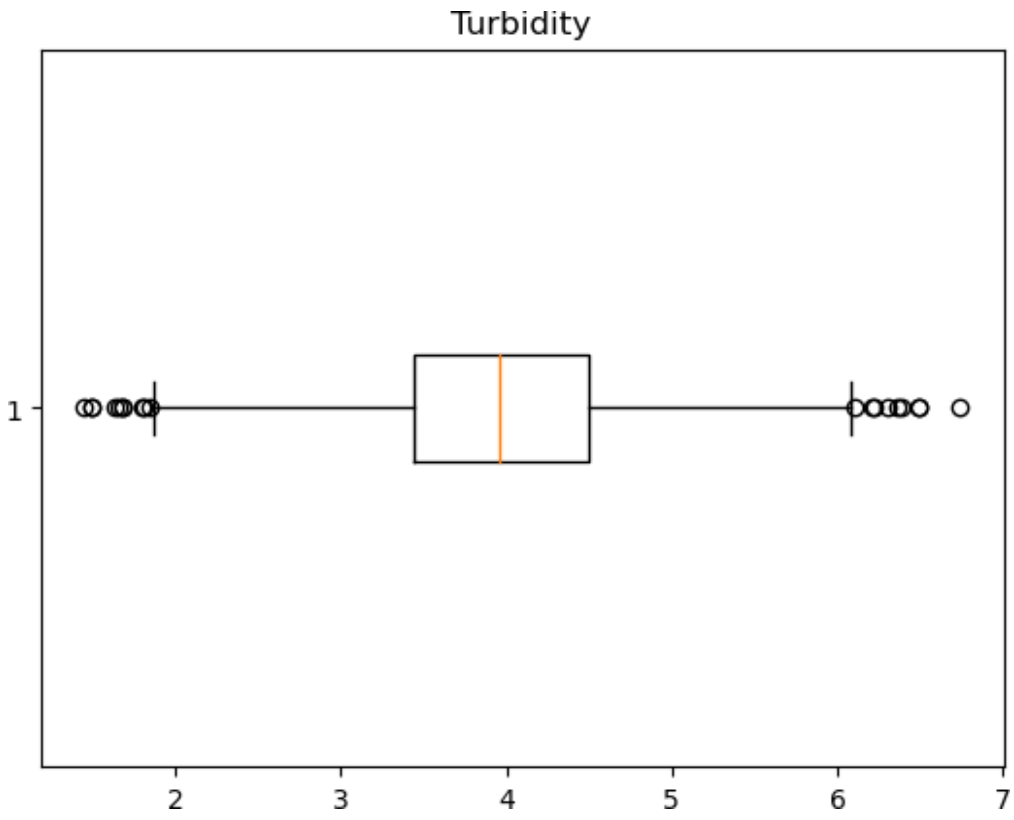

Chloramines
1
0
2
4
6
8
10
12

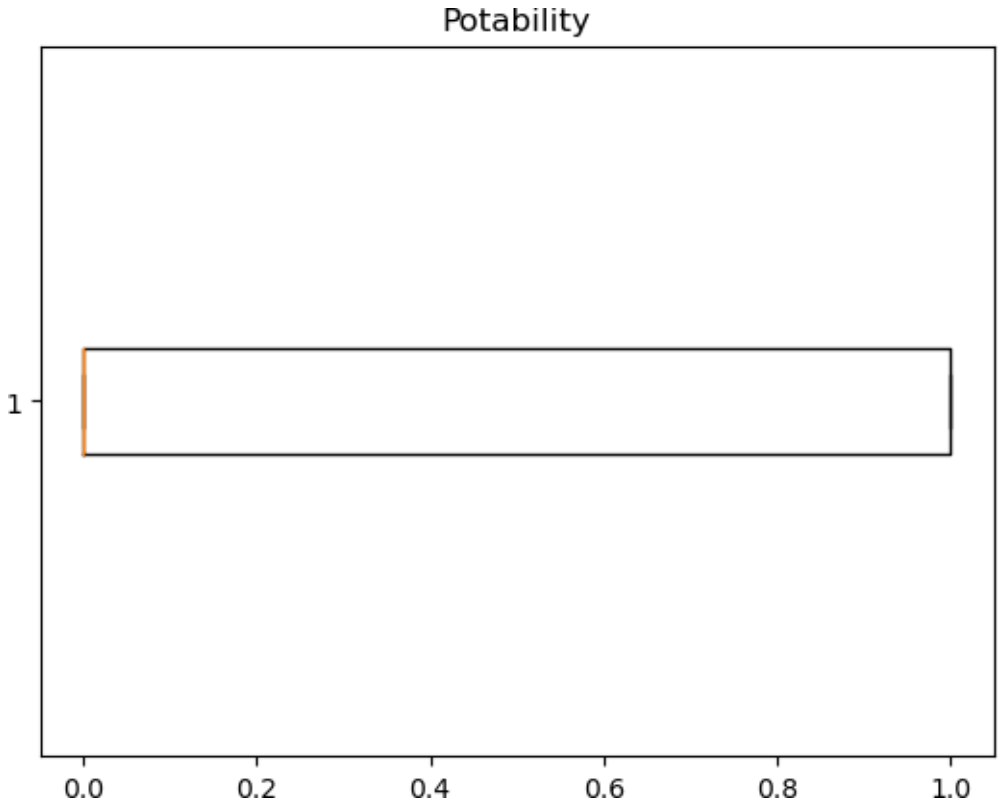

Sulfate
1
150
200
250
300
350
400
450


Conductivity
1
200
300
400
500
600
700


Organic_carbon
1
5
10
15
20
25

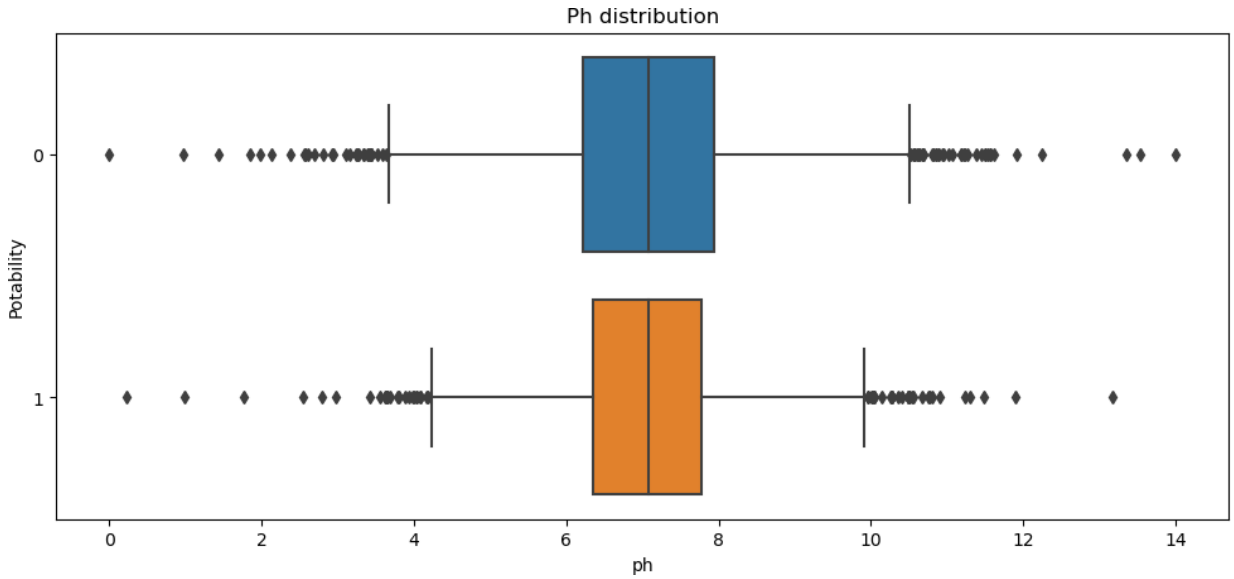

Trihalomethanes
1
0
20
40
60
80
100
120


Turbidity
1
2
3
4
5
6
7

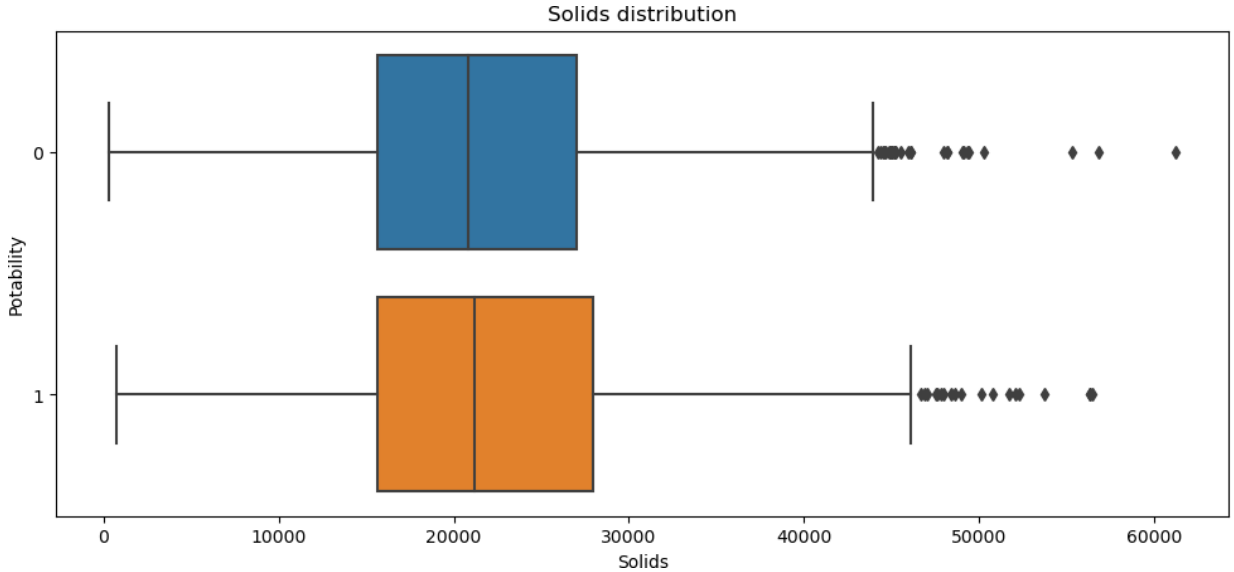

Potability
1
0.0
0.2
0.4
0.6
0.8
1.0

Checking for other relations

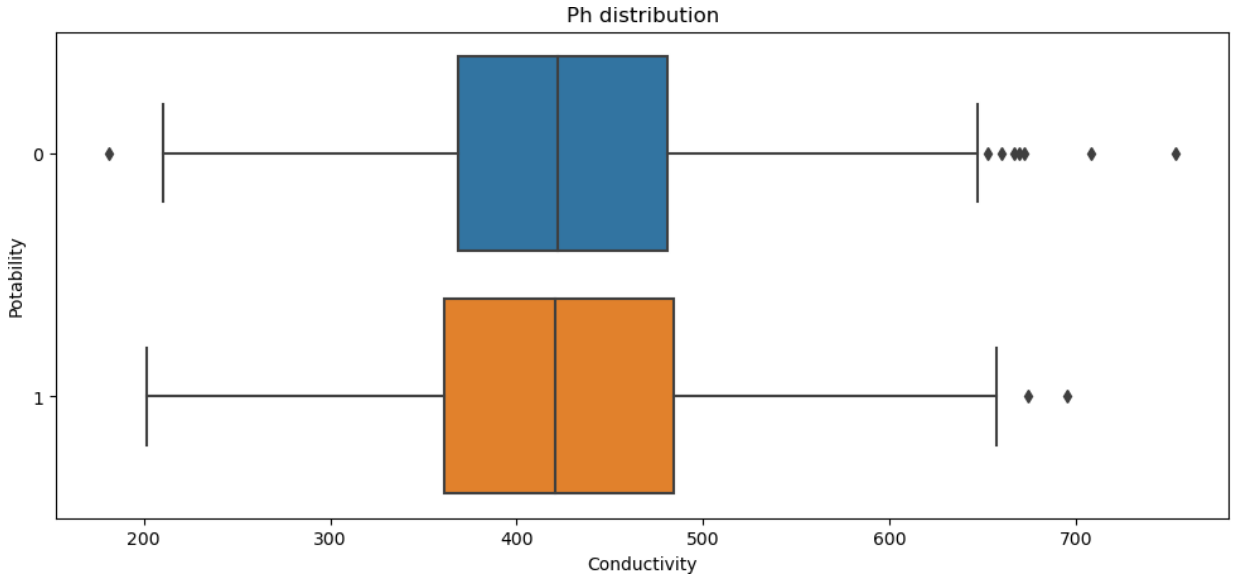
```
fig,ax  = plt.subplots(figsize = (12,5))
sns.boxplot(data =df, x = 'ph', y = 'Potability', orient =
'h').set(title = 'Ph distribution');
```




```
fig,ax  = plt.subplots(figsize = (12,5))
sns.boxplot(data =df, x = 'Solids', y = 'Potability', orient =
'h').set(title = 'Solids distribution');
```



```
fig,ax  = plt.subplots(figsize = (12,5))
sns.boxplot(data =df, x = 'Conductivity', y = 'Potability', orient =
'h').set(title = 'Ph distribution');
```



Phase_3 Conclusions

--> From the correlation heatmap plotted earlier, its clear that the pf level of the water and the hardness of the water are highly correlated.

>> The Outliers of each attribute in the dataset is properly visualized using boxplot

>> Sulfate has so many outliers as well as less correlated with most other attributes, thus it can be deleted if not needed.

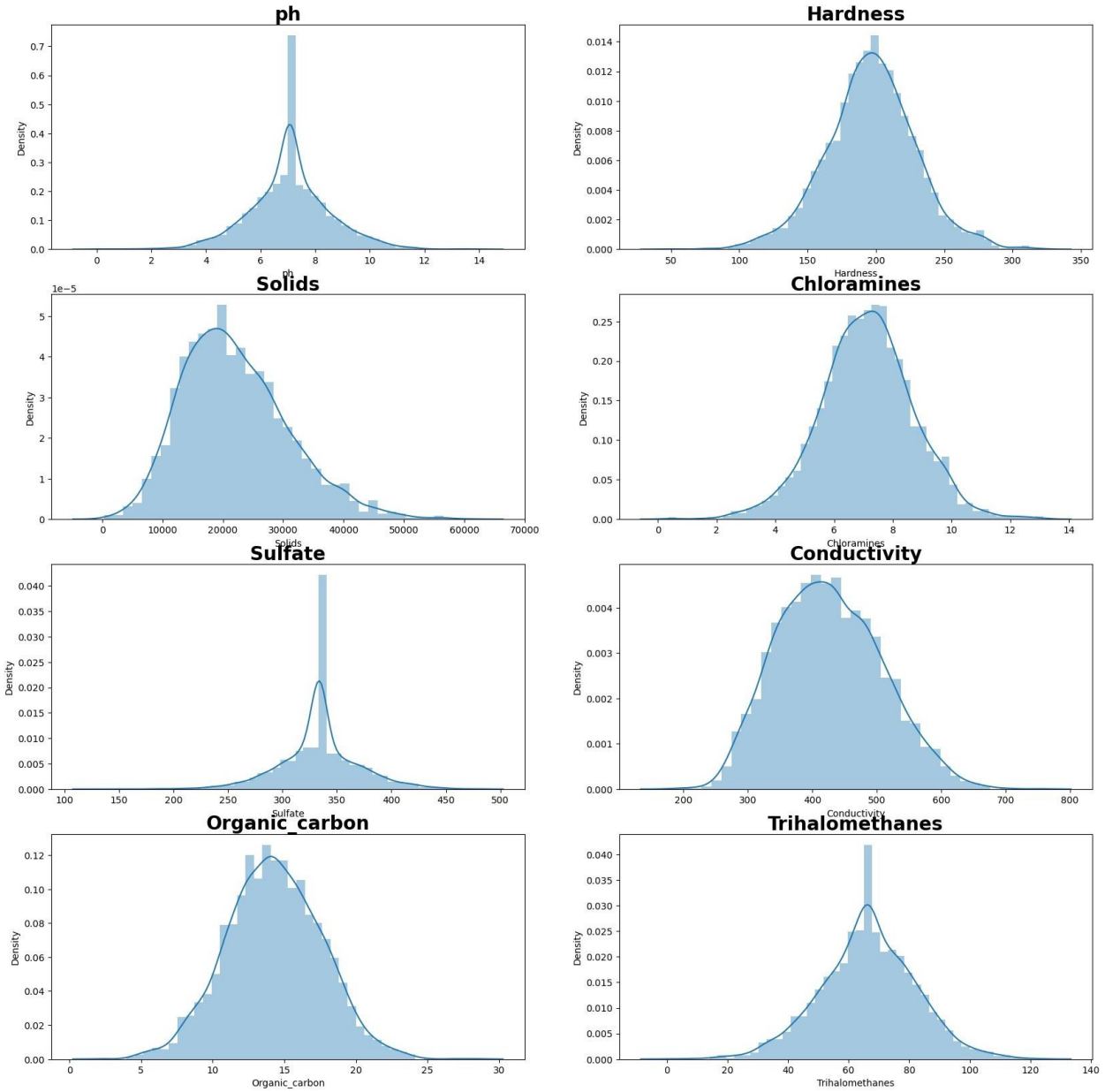
>> ph, Chloramine, solids also have many outliers

From other three comparative boxplot using ph and probability, it is clear that water which harmful for drinking and water which safe for drinking are almost slightly equally distributed in this samples

Phase_4

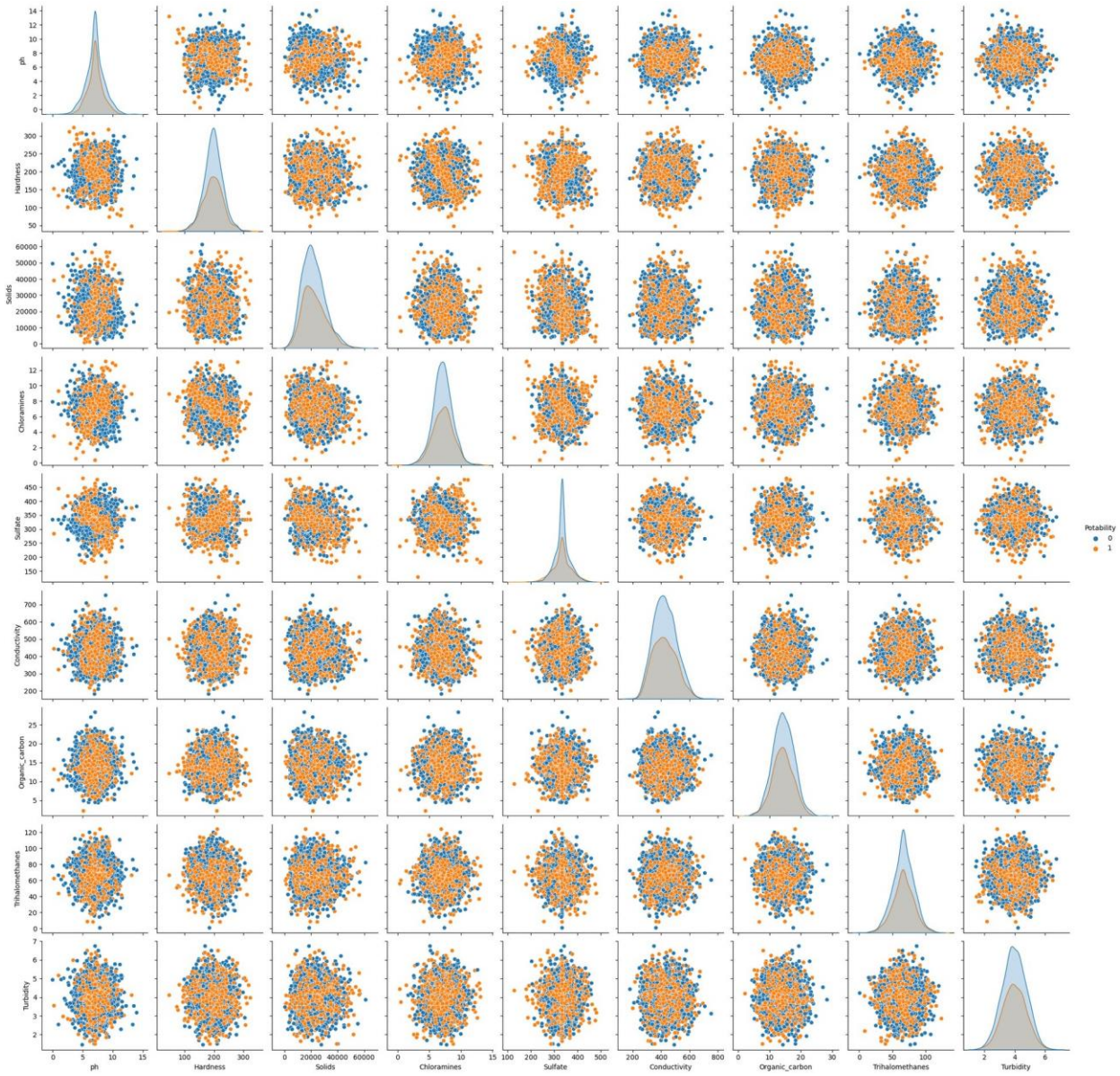
Feature Engineering

```
plt.figure(figsize=(20,20))
for i in range(8):
    plt.subplot(4,2,(i%8)+1)
    sns.distplot(df[df.columns[i]])

plt.title(df.columns[i],fontdict={'size':20,'weight':'bold'},pad=3)
plt.show()
```



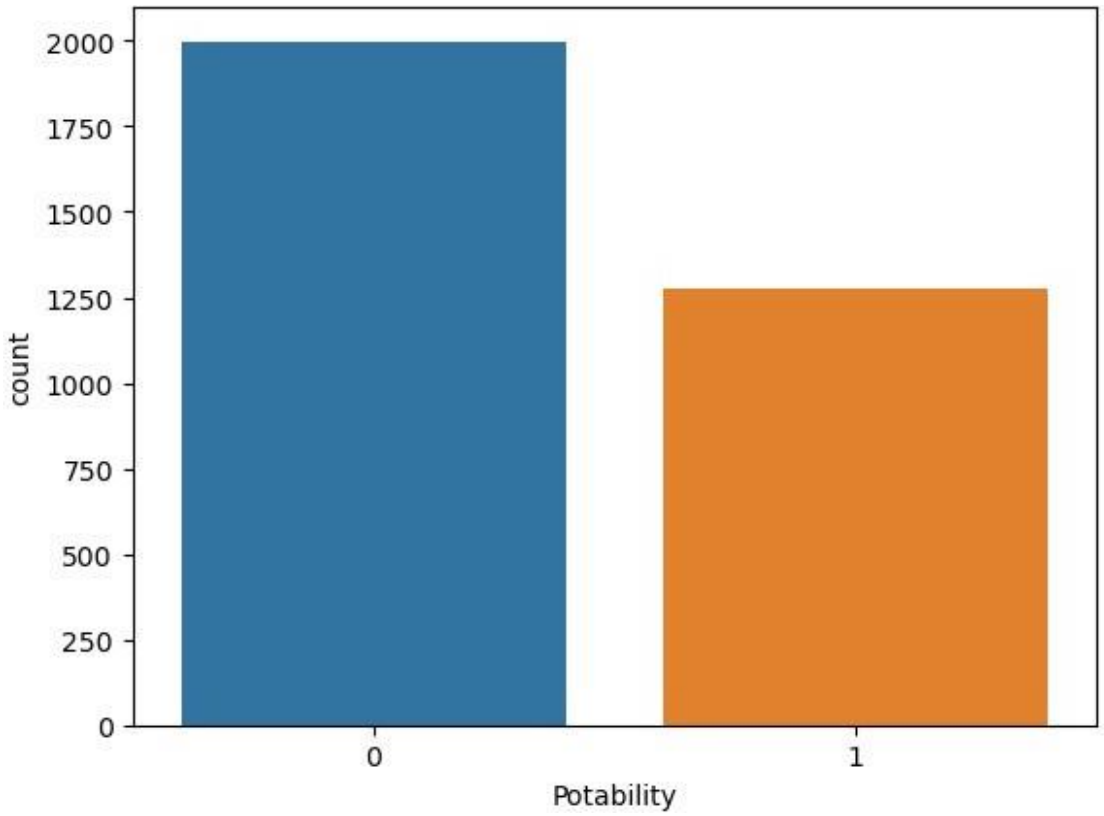

```
sns.pairplot(data=df, hue='Potability')
```

```
<seaborn.axisgrid.PairGrid at 0x1f75bf54b90>
```

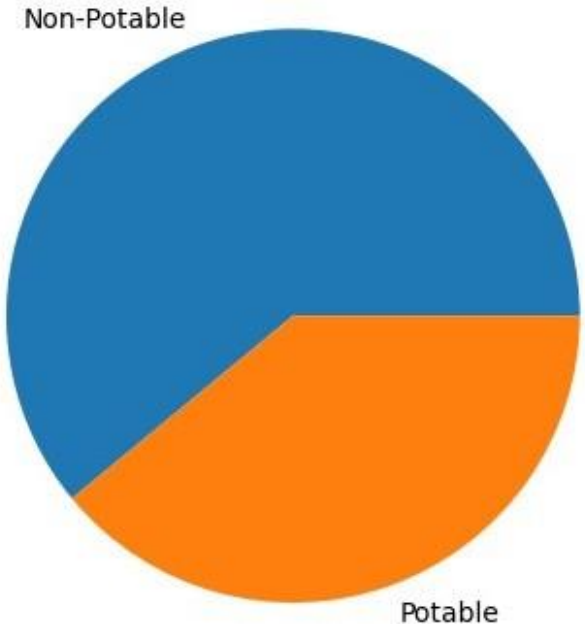



```
##Checking for distribution of Potable water

sns.countplot(x=df["Potability"])
```

```
<Axes: xlabel='Potability', ylabel='count'>
```




```
#Representing in a visually applealing pie chart

ratio = df.Potability.value_counts()
plt.pie(ratio, labels=['Non-Potable','Potable'])
plt.show()
```



MODEL Training and Evaluation

Seperating independent variable say X and dependent variable say Y

```
X =  df[['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate',
'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity']]
X.head()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity
0	7.080795	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180013	56.329076	4.500656
2	8.099124	224.236259	19909.541732	9.275884	333.775777	418.606213			
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516			
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813			

```
2        16.868637         66.420093    3.055934
3        18.436524        100.341674    4.628771
4        11.558279         31.997993    4.075075
```

```
y = df['Potability']
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: Potability, dtype: int64
```

Splitting the dataset into Train and Test for modeling

```
from sklearn.model_selection import train_test_split
#splitting the dataset

X_train,X_test,Y_train,Y_test =
train_test_split(X,y,test_size=.2,random_state=42)
```

Importing necessary libraries for modeling and Evaluating

```
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

log_reg = LogisticRegression()

dtc = DecisionTreeClassifier(criterion='entropy',max_depth=5)
```

Logistic Regression

```
log_reg.fit(X_train,Y_train)
tst2 = log_reg.predict(X_test)
```

Model accuracy

```
log_acc=accuracy_score(Y_test,tst2)

print("Train Set
```

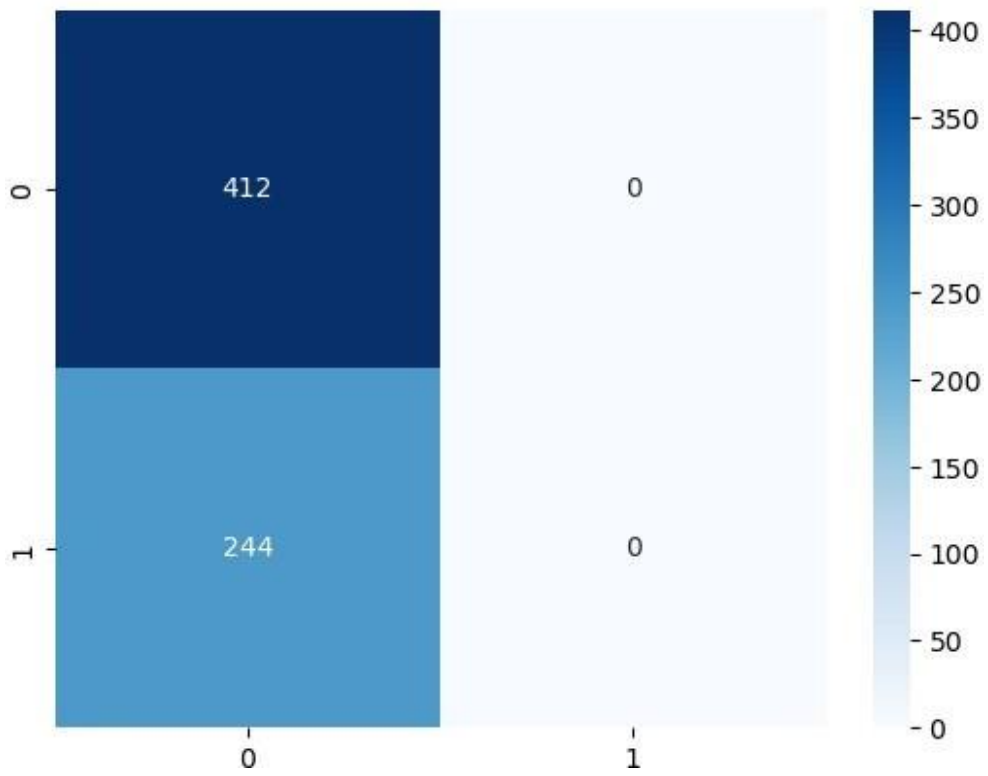
```
Accuracy:"+str(accuracy_score(Y_train,log_reg.predict(X_train))*100))
print("Test Set
Accuracy:"+str(accuracy_score(Y_test,log_reg.predict(X_test))*100))
```

```
Train Set Accuracy:60.57251908396947
Test Set Accuracy:62.80487804878049
```

Model Eevaluating

```
print('Logistic Regression\n')

log_cm = confusion_matrix(Y_test, tst2)
print(metrics.classification_report(Y_test, tst2))
sns.heatmap(log_cm, annot = True, fmt='d', cmap = 'Blues')
```

```
Logistic Regression

              precision    recall  f1-score   support

           0       0.63      1.00      0.77       412
           1       0.00      0.00      0.00       244

    accuracy                           0.63       656
   macro avg       0.31      0.50      0.39       656
weighted avg       0.39      0.63      0.48       656


<Axes: >
```



Decision Tree Classifier

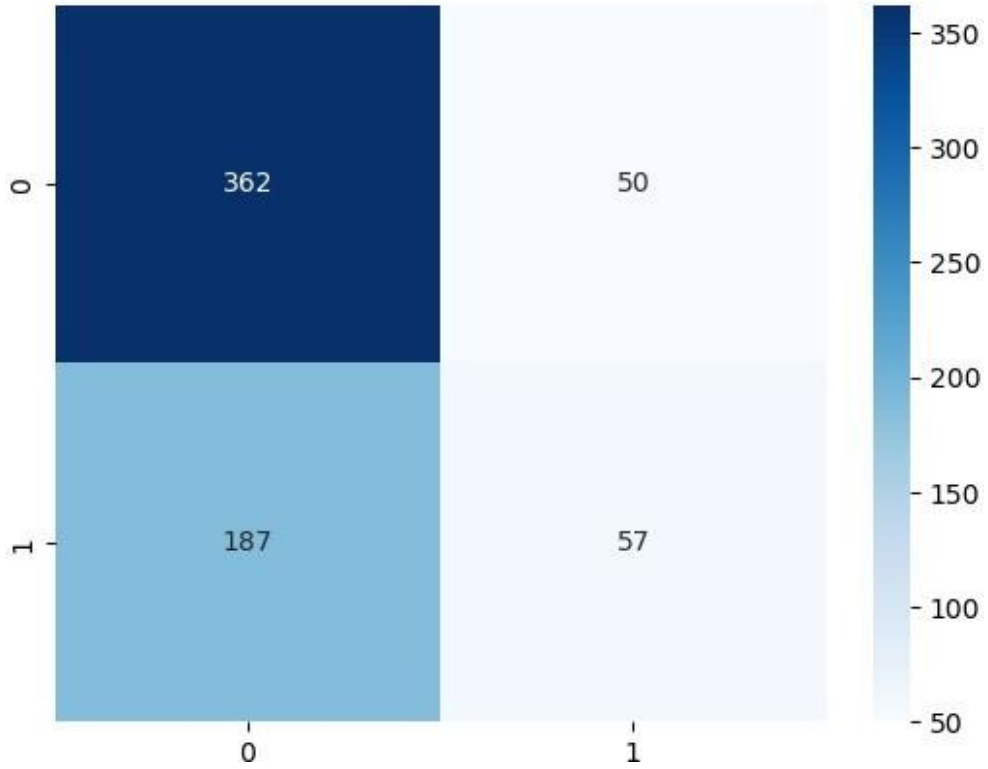
Model accuracy

```
dtc.fit(X_train, Y_train)
tst = dtc.predict(X_test)
```

Model Eevaluating

```
print('Decision Tree\n')

decision_tree_cm = confusion_matrix(Y_test, tst)
print(metrics.classification_report(Y_test, tst))
sns.heatmap(decision_tree_cm, annot = True, fmt='d', cmap = 'Blues')
```

```
Decision Tree

              precision    recall  f1-score   support

           0       0.66      0.88      0.75       412
           1       0.53      0.23      0.32       244

    accuracy                           0.64       656
   macro avg       0.60      0.56      0.54       656
weighted avg       0.61      0.64      0.59       656


<Axes: >
```



Executing Feature Engineering

Try removing columns with many outliers

```
## We found ph, chloramine,solids columns having many outliers... And
thus we train model without those data

X =  df[[ 'Hardness','Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity']]

X.head()
```

```
     Hardness      Sulfate  Conductivity  Organic_carbon
Trihalomethanes  \
0  204.890455  368.516441    564.308654       10.379783
86.990970
1  129.422921  333.775777    592.885359       15.180013
56.329076
2  224.236259  333.775777    418.606213       16.868637
66.420093
3  214.373394  356.886136    363.266516       18.436524
100.341674
4  181.101509  310.135738    398.410813       11.558279
31.997993

   Turbidity
0   2.963135
1   4.500656
2   3.055934
3   4.628771
4   4.075075
```

```
log_reg.fit(X_train,Y_train)
log_acc=accuracy_score(Y_test,log_reg.predict(X_test))


print("Train Set
Accuracy:"+str(accuracy_score(Y_train,log_reg.predict(X_train))*100))
print("Test Set
Accuracy:"+str(accuracy_score(Y_test,log_reg.predict(X_test))*100))
```

```
Train Set Accuracy:60.57251908396947
Test Set Accuracy:62.80487804878049
```

```
dtc.fit(X_train, Y_train)


dtc_acc= accuracy_score(Y_test,dtc.predict(X_test))

print("Train Set
Accuracy:"+str(accuracy_score(Y_train,dtc.predict(X_train))*100))
print("Test Set
Accuracy:"+str(accuracy_score(Y_test,dtc.predict(X_test))*100))
```

```
Train Set Accuracy:67.29007633587786
Test Set Accuracy:63.87195121951219
```

It seems to be same as the previous modeling

Phase_4 Conclusion

>> The two models Trained were Logistic regression model and Decision tree model

>> Out of the two models trained, *Decision Tree model* out performed Logistic Regression.

>> When we tried to improve the models by removing some columns which found to have many outliers and training the model again. this turned out the model to perform at same level.

>> From this move we can conclude that, from the given dataset, all the features or columns have same impact on the predictor variable and removing one thus slightly reduces the models performance.